FERMILAB-Conf-88/43

Design, Implementation, and Operation of a Class Based Batch Queue Scheduler for VAX/VMS*

Keith Chadwick
Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

May 20, 1988

*To be presented at the DECUS, Spring, 1988 Symposium, Cincinnati, Ohio, May 16-20, 1988

Design, Implementation, and Operation of a Class Based Batch Queue Scheduler for VAX/VMS

Keith Chadwick
Fermi National Accelerator Laboratory[1]
Batavia, IL 60510

Abstract

Fermilab found that the standard VMS batch configuration options were inadequate for the job mix that exists on the Fermilab central computer facility VAX cluster. Accordingly, Fermilab designed and implemented a class based batch queue scheduler.

This scheduler makes use of the standard VMS job controller and batch system. Users interact with the scheduler at job submission time by specification of CPU time limits and batch job characteristics. This scheduler allows Fermilab to make efficient use of our large heterogeneous VAX cluster which contains machines ranging from a VAX 780 to a VAX 8800.

The scheduler was implemented using the VMS system services $GETQUI and $SNDJBC, without changes to the existing VMS job scheduler. As a result, the scheduler should remain compatible with future VMS versions.

This session will discuss the design goals, implementation, and operational experience with Fermilab's class based batch queue scheduler.

Introduction

The introduction by Digital of the VAX cluster has had an enormous impact on VAX users and managers. Several thousand VAX clusters have been installed and are in operation today. More VAX clusters are being installed daily and existing VAX clusters are growing larger and more complex. Existing VAX clusters rival traditional mainframe computers in aggregate compute power and disk resources. With this increasing power and complexity, greater and greater demands are placed on the standard VAX/VMS batch job scheduler. In large VAX clusters, the aggregate number of batch queues can quickly overwhelm the unsophisticated user and even the experienced system manager.

Fermilab in late 1986 was rapidly approaching the point of overwhelming complexity and diminishing returns using the "standard" Digital VAX cluster batch configuration strategies. An assessment of the problem revealed that a radical departure from the usual VAX cluster batch configuration options would be required in order to flexibly support the projected growth in demand for batch computing cycles at Fermilab.

The Fermilab Environment

The Fermilab central computing facility (FNAL) VAX cluster currently has the following hardware:

- Processors: 1 VAX 8800, 2 VAX 8650's, 1 VAX 8600, 2 VAX 785's, and 1 VAX 780.
- Hsc's: 2 HSC70's, and 2 HSC50's.
- Disks: 40 RA81's, and 12 RA82's.
- Tapes: 14 TA78's.

The user community currently numbers approximately 2800. Typical peak VAX cluster wide interactive loads range from 250 to 280 simultaneous users (90 to 100 per 8650, and 70 to 80 on the 8600). The remaining processors; the 8800, the two 785's, and the 780 are used exclusively for batch processing. Batch processing is also run on the 8650's and the 8600 during non-prime hours.

Prior to the implementation of the batch queue scheduler, Fermilab's VAX cluster batch queues were divided into the five general "classes" described below:

[1]Fermilab is operated under contract to the United States Department of Energy. This material was prepared as a part of work sponsored by the United States Department of Energy. The Department of Energy, Universities Research Association, Inc., and their agents and employees, make no warranty, express or implied, and assume no legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, nor represent that its use would not infringe privately owned rights.

- SYSTEM Job - Existed to perform system management functions, housekeeping, backups, accounting reports, etc.
- STANDBY Job - Existed to prevent cycles being lost to the null process. Jobs in this class are run at a very low priority, and typically involve calculations (such as lattice gauge theory) which require several days of CPU time.
- LONG Job - Existed to support long (many hours) jobs that are generally CPU bound. Jobs in this class are run at a low priority, and typically involve calculations (such as the large majority of "production" analyses) which require on order one day of CPU time.
- MEDIUM Job - Existed to support moderate length jobs that can be either CPU bound, or I/O bound but that will complete in a finite amount of time. Jobs in this class are run at a moderate priority, and typically involve calculations (such as creation of object libraries, or initial debugging of analyses) which require at most four hours of CPU time.
- SHORT Job - Existed to reduce the interactive load, by providing users with a pseudo-interactive turn around in a batch job. Jobs in this class are run at a relatively high priority, and typically involve short calculations (such as compiles and links) which require at most thirty minutes of CPU time.

The implementation of this batch system was straight out of Digital's *Guide to VAX/VMS System Management and Daily Operations* and *Guide to VAXclusters*: One generic queue together with the associated executor queues on each processor per job class. In addition to the above general classes, there also existed special batch queues (again with a generic and associated executor queues on each processor) which were intended to provide additional CPU cycles to specific users and/or groups.

The large number of generic and executor queues (over 35 just for the general classes) resulted in a significant user confusion as to the appropriate batch queues for their batch jobs. The user would first have to estimate the CPU requirements, then decide upon an appropriate generic queue (some users liked to "jump" over the generic queue and submit directly into the executor queues), submit their job, and wait for the results. Typically users were confused by the myriad available job classes, generic queues, and executor queues.

In addition to the user confusion, the Fermilab environment is somewhat unusual in that laboratory personnel and their university collaborators may move from assignment to assignment, or may simultaneously have multiple assignments, during their stay at the laboratory. The standard VMS grouping scheme would require that the Fermilab VAX cluster management repeatedly modify the user's UIC in order to track these changes, or issue multiple accounts to individual users on a continuing basis. This would in turn require that the user's file base be repeatedly modified to assure proper file ownership. Fortunately rights identifiers and access control lists were introduced with VMS V4.0. The use of these tools has allowed Fermilab to track the current assignments by granting and revoking the appropriate rights identifiers. Unfortunately this flexibility at the file base level was not matched in the batch queues; VMS did not allow the use of rights identifiers and access control lists with batch queues.

Design Goals

The model used for the establishment of the design goals for the batch queue scheduler was based on Fermilab's existing batch queue organization. Based on this model, numerous design goals were established for the batch queue scheduler prior to the start of the actual implementation of the software. These design goals were repeatedly examined and reviewed during the implementation cycle and revised as necessary and appropriate. This SASD methodology resulted in the following list of design goals:

- Implement the batch queue scheduler in a familiar high level language. The available programmer expertise at Fermilab resulted in the following languages being considered: VAX Macro, VAX Basic, VAX C, VAX Pascal, and VAX Fortran. VAX Fortran was finally chosen as the implementation language. The implementation of the queue scheduler was highly modular with individual subroutines having well defined inputs, actions, and outputs. All source code, together with the modifications, has been tracked with Digital's Code Management System DEC/CMS.
- Implement the batch queue scheduler without recourse to user written kernal mode code or modification to the standard VMS job controller. The batch queue scheduler interface to the VAX/VMS job controller must only use documented systems services and run time library routines. This requirement basically serves to ensure upward compatibility with future versions of VAX/VMS.
- Sort batch jobs into classes based on the CPU time limit and characteristics specified at the time of the job submission, together with the user's name, group, and/or right's identifiers held.
- Break down classes into four class types:
  - User
  - Group/Charge Code
  - (Rights) Identifier
  - Generic
- Reduce the total number of generic and executor batch queues (ideally to a single VAX cluster wide generic queue, and a single executor queue per processor). Within executor queues, job "slots" are reserved

(pre-allocated) for batch jobs meeting the previously defined classes. Support the allocation of the number of jobs per processor proportionally to the relative CPU power of the processor.

- Eliminate direct user access to the executor queues, and require users to submit their jobs into the generic queue. Users are allowed to select or differentiate processors via previously defined queue characteristics.
- Allow management to restrict user access to other user jobs such that a user may only examine the batch queue enties of their jobs (or jobs submitted by members of the same group/charge code).
- Initiate batch jobs at the base priority appropriate to the batch job class.
- Support reasonable job defaults:
  - CPU time = Infinite.
  - No tape drives required.
  - Working set = Executor queue working set.
- Require users to specify their jobs tape, CPU, and memory requirements if these requirements are different that the default. The standard SUBMIT qualifiers /CPUTIME, /WSDEFAULT, /WSEXTENT, and /WSQUOTA are used to inform the queue scheduler of the CPU time and memory requirements. Queue characteristics ONE_TAPE, TWO_TAPE, THREE_TAPE, and FOUR_TAPE are used to signal the requirements of one, two, three, and four tape drives respectively.
- Support the specification of CPU time limits in a form independent of the actual executor queue chosen to run the job. Executor queues are "rated" according to the speed of the host processor.
- Support optional user notification (via a VAX cluster wide broadcast) whenever a user's batch jobs enter an executor queue (only if the user specifies /NOTIFY with the batch job).
- Support the use of rights identifiers, usernames, and groups to reserve slots for the defined classes in executor queues.
- Allow modification of the batch job mix without halting and restarting of the queue scheduler.
- Support multiple copies of the queue scheduler in order to allow for automatic failover in the case where the node that is running the active queue scheduler crashes.

Implementation

Keeping the above list of design goals in mind, the following is a general description of the queue scheduler:

- The active batch queue scheduler establishes a mutex lock, so that the alternate batch queue schedulers do not attempt to schedule the same batch queues.
- The queue scheduler then reads in the batch queue job mix parameter file (this file is read in once per cycle of the queue scheduler to allow modification of the job mix on the "fly"). This file contains the following information (an example parameter file is included in the appendix):
  - The name of the generic queue to schedule, the default CPU time, and the maximum number of jobs that a single user may have in execution simultaneously. Example:

    ```
    Generic_Queue <queue_name> -
      /Default_Cpu_Limit=<delta_time> -
      /Job_Limit=#
    ```

  - The class definitions (class name, CPU limit, base priority, class type, the total number of jobs which a single user may have in execution simultaneously under this class, and a user/group/charge code/rights identifier name if appropriate). Example:

    ```
    Define_Class <class_name> -
      /Cpu_Limit=<delta_time> -
      /Priority=# -
        /User=<username> -
        /Group=<charge_code> -
        /Identifier=<string> -
        /Generic -
      /[No]Alternate -
      /Job_Limit=#
    ```

  - The names of the executor queues to schedule jobs into from the generic queue, the relative CPU power of the host processor, and the total number of job slots in the executor queue. Example:

    ```
    Executor_Queue <queue_name> -
      /Relative_CPU_Power=#.# -
      /Total_Slots=#
    ```

  - The slot reservations on a class by class basis on each executor queue. Example:

    ```
    Reserve_Slots <queue_name> -
      /Class=<class_name> -
      /Job_Limit=#
    ```

- The queue scheduler examines (using the $GETQUI system service) the executor queues that it is scheduling to determine the appropriate parameters.
  - JOB_LIMIT

    - WSDEFAULT
    - WSEXTENT
    - WSQUOTA
    - CPU_DEFAULT
    - CPU_MAXIMUM
    - CHARACTERISTICS

- The queue scheduler examines (using the $GETQUI system service) the executor queues that it is scheduling to determine if there are any open job slots.
- The queue scheduler examines (using the $GETQUI system service) the generic queue to determine if there exists any pending jobs for the open job slots.
- If a pending job requires CPU time greater than any open job slot, then it is passed over.
- If a pending job requires a working set larger than the working set of any executor queue with any open job slot, then it is passed over. Note that the working set of the job slot is associated with the executor queue in which the job slot resides.
- If a pending job requires tape drives, and there are insufficient tape drives available, then it is passed over.
- If a open job slot is reserved for an account/charge code/rights identifier, and the pending job does not posses the appropriate account/charge code/rights identifier, then it is passed over.
- If a pending job satisfies ALL the requirements of an open job slot, then it is copied into the open job slot (via the $SNDJBC system service) in the target executor queue. If the user specified the CPU time required, then the CPU time specified must be converted from "generic units" (typically VAX 11/780 units) into the target executor queue processor specific units (*i.e.* the user specified CPU time is divided by the relative CPU power). The batch queue scheduler modifies the parameters of the target executor queue, which allows the job to begin execution. Once the job has begun execution, the target executor queue parameters are restored to the default configuration (refer to section below on Problems for further details)
- After moving a pending job into the open job slot, the queue scheduler issues a VAX cluster wide broadcast (via the $BRKTHRU system service) to the user (if the user submitted the batch job with the /NOTIFY qualifier) in order to notify them that their job has begun execution in the target executor queue.
- Finally the queue scheduler sleeps for 60 seconds, and then resumes its task loop.

Problems

As one might expect with a project of this magnitude, numerous minor problems and curiosities were encountered during the implementation of Fermilab's batch queue scheduler:

- The lack of VAX cluster clock synchronization ("fixed" in VMS V4.6 with the addition of SET TIME/CLUSTER).
- The generic batch queue must not have a list of target executor queues, and the qualifier /NOGENERIC is required on all executor queues in order to prevent the standard VAX/VMS job scheduler scheduling batch jobs into arbitrary executor queues.
- The inability of $CHECK_ACCESS system service to support simple third party mode tests via the addition of the rights identifier BATCH to a users default priv's (*i.e.* if the job is executing in batch, can user X access tape drive N).
- The inability of a privileged user or application to force VMS to start execution of a user's batch job in a batch queue to which the user does not possess write access (solved by "toggling" the world write protection bit on the target executor queue after coping the user's job into the queue - not a very elegant solution).

VMS Enhancement Wish List

While implementing the class based batch queue scheduler, the following enhancements to the standard VAX/VMS job scheduler would have greatly simplified the task:

- A mechanism whereby the queue scheduler could receive AST notification from the job scheduler for events of interest:
    - Submission of a job into the SYS$BATCH generic queue.
    - Completion of a job in an executor queue.

  A general solution might be to implement a new qualifier /SCHEDULER=*file-name* for the INITIALIZE/QUEUE command. This qualifier would allow users to specify an alternative job scheduler for specific batch queues in a similar manner as the /PROCESSOR qualifier allows users to specify their own print symbionts.
- Allow users to specify the qualifier /OPERATOR=*string* when submitting batch jobs as well as print jobs. An example of a possible application in the Fermilab environment would be for a user to specify the volume labels of the magnetic tapes that the job will require via such an operator comment.

Operation

Fermilab has operated the class based batch queue scheduler since December 1986:

- Version 1.0 was put into operation in December 1986. For the next 8 months the queue scheduler was shaken down on the SYS$BATCH queue, and the remaining generic queues (SYS$STANDBY, SYS$LONG, SYS$SHORT, etc.) continued operation in parallel using the standard VMS batch queue scheduling algorithm.

- Version 2.0 was installed in August 1987. class definitions corresponding to the existing generic queues were placed in the queue scheduler parameter file, and the obsolete generic queues were deleted from the Fermilab VAX cluster.

    - The qualifier /IDENTIFIER on the DEFINE_CLASS verb was enhanced to allow the presence of an optional string /IDENTIFIER=*string*. If the string was not present, the name of the class was used as the rights identifier and batch queue characteristic which defined the class. If the string was present, then that string was used instead of the name of the class. This enhancement allows multiple queue classes to be defined without requiring multiple rights identifiers.

    - The mechanism within the queue scheduler for determining the number of available tape drives for a given user was extended, in order to support the use of access control lists (ACL's) on tape drives. The ACL's were placed on selected tape drives in order to preallocate them for specific accounts and/or groups.

- Version 2.1 was installed in October 1987 with the following enhancements:

    - The job classification mechanism within the queue scheduler was extensively modified in order to generate a list of job classes which match the parameters of the batch job. Prior versions of the queue scheduler only matched a batch job to at most two classes (one reserved and one generic class). Version 2.1 of the queue scheduler generates a class search list of job classes (said class search list may contain from 0 to N elements, where N is less than or equal to the number of currently defined job classes). The class of the batch job will be defined as the first entry in the class search list for which there is an open job slot.

    - An optional qualifier /[NO]ALTERNATE was added to the DEFINE_CLASS verb within the queue scheduler parameter file. Specification of the /NOALTERNATE qualifier for the DEFINE_CLASS verb results in the class search list, for batch jobs which satisfy the requirements of the aforementioned job class, containing one and only one entry. The default value of this qualifier is /ALTERNATE. The use of the /NOALTERNATE qualifier is restricted to very special job classes (such as SYSTEM).

    - An optional qualifier /DEFAULT_CPU_LIMIT=*delta_time* was added to the verb GENERIC_QUEUE within the queue scheduler parameter file. If this qualifier is not specified the default value is /DEFAULT_CPU_LIMIT=*Infinite*.

    - The mutex logic which prevented multiple copies of the queue scheduler from simultaneously attempting to schedule jobs from a single generic batch queue was changed to use the distributed lock management system services. Previous versions of the queue scheduler used a rather primitive interlocking mechanism via FORTRAN open statements.

    - The verb EXECUTOR_QUEUE was created as an alternate form of the DEFINE_QUEUE verb within the queue scheduler parameter file.

    - The wake up interval was changed from a fixed delta time (typical value sixty seconds), to a value randomly generated in the interval:

```
[0.5*Wake_Up_Interval,1.5*Wake_Up_Interval]
```

    Note that the mean value will be equal to the fixed wake up interval. This change was undertaken to avoid possible phase lock interference with user processes (an example of a possible interference problem would be the inability for a user to allocate tape drives since they and the queue scheduler both "wake up" at one minute intervals).

    - The queue scheduler debug reports were extensively reformatted in order to better display the large number of queue scheduler queue classes currently defined on the Fermilab VAX cluster.

- Version 2.2 was installed in April 1988 with the following enhancements:

    - A lock value block was added to the lock block passed to the distributed lock manager in order to allow the non-active queue schedulers to determine the identification of the active queue scheduler without resort to the $GETLKI system service.

    - An optional qualifier /JOB_LIMIT was added to the GENERIC_QUEUE and the DEFINE_CLASS verbs in the queue scheduler parameter file. This qualifier was added to support the implementation of user job limits. These job limits are limits on the total number of simultaneous jobs (under all classes, and on the number of jobs within a specific class respectively) that an individual user may have in execution on the system. If this qualifier is not specified, the default value is /JOB_LIMIT=*Infinite*.

- The string *Default* was added as a valid value for the /CPU_LIMIT qualifier on the DEFINE_CLASS command within the queue scheduler parameter file. The value used for *Default* is the value present on the /DEFAULT_CPU_LIMIT qualifier on the GENERIC_QUEUE verb.
- The queue scheduler was changed to use the $GETQUI system service to determine the characteristic number associated with the characteristics ONE_TAPE, TWO_TAPE, THREE_TAPE, and FOUR_TAPE, rather than using hard coded parameter values.
- A bug which resulted in the job classification algorithm being unable to correctly determine the class of an already executing batch job was corrected. This bug only manifested itself under extremely rare circumstances and configurations.

Summary

In general, Fermilab's experience with the queue scheduler has been very satisfactory, both from a systems management viewpoint, and also from a users viewpoint. The flexible configuration, together with the single point of control provided by the queue scheduler parameter file has repeatedly proven that the investment in the development and support of the queue scheduler has been repaid handsomely. The users environment has been greatly simplified, since the large number of generic queues and executor queues has been reduced to a single VAX cluster wide generic queue, together with one executor queue on each VAX cluster processor. Turn around time is good, even when large numbers of computationally bound jobs are in execution and pending in the generic queue (users are quite happy to see their five minute compilation job leapfrog dozens of long computation jobs pending in the SYS$BATCH generic queue and "immediately" enter execution). From a systems support viewpoint, the simplification of the users environment has resulted in a corresponding decrease in the number of user complaints and problems related to the batch queues. The ability to specify CPU time limits (in a processor independent fashion), memory, and tape drive requirements has allowed the Fermilab VAX cluster management and the users to better monitor and make use of the resources of Fermilab's VAX cluster.

Acknowledgements

I wish to thank David J. Ritchie, Steven J. Kalisz, and Chip Kaliher of the Fermilab DEC Systems Group for their contributions to the discussions which provided the basis for the design of Fermilab's batch queue scheduler.

References

Digital Equipment Corporation. *Guide to VAX/VMS System Management and Daily Operations*, Version 4.0, September 1984, Order number AA-Y507A-TE.

Digital Equipment Corporation. *Guide to VAXclusters*, Version 4.0, September 1984, Order number AA-Y513A-TE.

Digital Equipment Corporation. *VAX/VMS DCL Dictionary*, Version 4.4, April 1986, Order number AA-Z200C-TE.

Digital Equipment Corporation. *VAX/VMS System Services Reference Manual*, Version 4.4, April 1986, Order numbers AA-Z501B-TE, AD-Z501B-T1.

Appendix - Example Parameter File

```
Generic_Queue SYS$BATCH /Default_Cpu_Limit=Infinite /Job_Limit=10

Define_Class System         /Cpu_Limit=Infinite /Priority=4 /User       /NoAlternate
Define_Class CDF_Short_Job  /Cpu_Limit=00:30:00 /Priority=4 /Identifier=CDF_Experiment /Job_Limit=4
Define_Class CDF_Medium_Job /Cpu_Limit=04:00:00 /Priority=3 /Identifier=CDF_Experiment /Job_Limit=4
Define_Class CDF_Long_Job   /Cpu_Limit=Infinite /Priority=2 /Identifier=CDF_Experiment /Job_Limit=2
Define_Class CDF_Reduction  /Cpu_Limit=Infinite /Priority=5 /Identifier /NoAlternate   /Job_Limit=1
Define_Class CDF_Server     /Cpu_Limit=Infinite /Priority=4 /User       /NoAlternate   /Job_Limit=1
Define_Class E687_C         /Cpu_Limit=Infinite /Priority=2 /Group                     /Job_Limit=1
Define_Class Short_Job      /Cpu_Limit=00:30:00 /Priority=4 /Generic                   /Job_Limit=4
Define_Class Medium_Job     /Cpu_Limit=04:00:00 /Priority=3 /Generic                   /Job_Limit=4
Define_Class Long_Job       /Cpu_Limit=Infinite /Priority=2 /Generic                   /Job_Limit=4

Executor_Queue FNALA_BATCH /Relative_CPU_Power=1.5 /Total_Slots=7
Executor_Queue FNALB_BATCH /Relative_CPU_Power=5.5 /Total_Slots=10
Executor_Queue FNALC_BATCH /Relative_CPU_Power=5.5 /Total_Slots=10
Executor_Queue FNALD_BATCH /Relative_CPU_Power=1.5 /Total_Slots=7
Executor_Queue FNALE_BATCH /Relative_CPU_Power=4.0 /Total_Slots=15
Executor_Queue FNALF_BATCH /Relative_CPU_Power=5.5 /Total_Slots=24
Executor_Queue FNALG_BATCH /Relative_CPU_Power=1.0 /Total_Slots=5

Reserve_Slots FNALA_BATCH /Class=System         /Job_Limit=2
Reserve_Slots FNALA_BATCH /Class=Short_Job      /Job_Limit=1
Reserve_Slots FNALA_BATCH /Class=CDF_Short_Job  /Job_Limit=1
Reserve_Slots FNALB_BATCH /Class=System         /Job_Limit=2
Reserve_Slots FNALB_BATCH /Class=Medium_Job     /Job_Limit=2
Reserve_Slots FNALB_BATCH /Class=CDF_Medium_Job /Job_Limit=2
Reserve_Slots FNALC_BATCH /Class=System         /Job_Limit=2
Reserve_Slots FNALC_BATCH /Class=Long_Job       /Job_Limit=1
Reserve_Slots FNALC_BATCH /Class=CDF_Long_Job   /Job_Limit=1
Reserve_Slots FNALD_BATCH /Class=System         /Job_Limit=2
Reserve_Slots FNALD_BATCH /Class=E687_C         /Job_Limit=2
Reserve_Slots FNALE_BATCH /Class=System         /Job_Limit=2
Reserve_Slots FNALE_BATCH /Class=CDF_Server     /Job_Limit=1
Reserve_Slots FNALF_BATCH /Class=System         /Job_Limit=2
Reserve_Slots FNALF_BATCH /Class=Short_Job      /Job_Limit=2
Reserve_Slots FNALF_BATCH /Class=CDF_Short_Job  /Job_Limit=2
Reserve_Slots FNALF_BATCH /Class=Medium_Job     /Job_Limit=2
Reserve_Slots FNALF_BATCH /Class=CDF_Medium_Job /Job_Limit=2
Reserve_Slots FNALF_BATCH /Class=Long_Job       /Job_Limit=3
Reserve_Slots FNALF_BATCH /Class=CDF_Long_Job   /Job_Limit=3
Reserve_Slots FNALF_BATCH /Class=CDF_Reduction  /Job_Limit=1
```